In [3]:
```
import pandas as pd
import os
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

In [4]:
```
#Changing Directory
[redacted]
```

In [5]:
```
#Reading in the files
train_data= pd.read_csv("train_values.csv", index_col= 'building_id')
train_labels= pd.read_csv("train_labels.csv", index_col= 'building_id')
test_data= pd.read_csv("test_values.csv", index_col= "building_id")
```

In [6]:
```
# Checking shape and size of the data
print('training data shape: ', train_data.shape, '\ntest data shape: ', te
st_data.shape, '\nTraining Label Shape: ', train_labels.shape)
```

```
training data shape:  (260601, 38)
test data shape:  (86868, 38)
Training Label Shape:  (260601, 1)
```

In [7]:
```
#Let's take a look at first few values of training data
train_data.head()
```

Out[7]:

	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area_percentage
building_id						
802906	6	487	12198	2	30	6
28830	8	900	2812	2	10	8
94947	21	363	8973	2	10	5
590882	22	418	10694	2	10	6
201944	11	131	1488	3	30	8

5 rows × 38 columns

In [8]:
```
#Let's take a look at our labels
train_labels.head()
```

Out[8]:

	damage_grade
building_id	
802906	3
28830	2

94947	3
590882	2
201944	3

In [4]:
```
#Defining train and label
train_label= train_labels[['damage_grade']]
```

In [9]:
```
#Let's take a look if Training Data has any missing values
train_data.isnull().sum()
```

Out[9]:
```
geo_level_1_id                            0
geo_level_2_id                            0
geo_level_3_id                            0
count_floors_pre_eq                       0
age                                       0
area_percentage                           0
height_percentage                         0
land_surface_condition                    0
foundation_type                           0
roof_type                                 0
ground_floor_type                         0
other_floor_type                          0
position                                  0
plan_configuration                        0
has_superstructure_adobe_mud              0
has_superstructure_mud_mortar_stone       0
has_superstructure_stone_flag             0
has_superstructure_cement_mortar_stone    0
has_superstructure_mud_mortar_brick       0
has_superstructure_cement_mortar_brick    0
has_superstructure_timber                 0
has_superstructure_bamboo                 0
has_superstructure_rc_non_engineered      0
has_superstructure_rc_engineered          0
has_superstructure_other                  0
legal_ownership_status                    0
count_families                            0
has_secondary_use                         0
has_secondary_use_agriculture             0
has_secondary_use_hotel                   0
has_secondary_use_rental                  0
has_secondary_use_institution             0
has_secondary_use_school                  0
has_secondary_use_industry                0
has_secondary_use_health_post             0
has_secondary_use_gov_office              0
has_secondary_use_use_police              0
has_secondary_use_other                   0
dtype: int64
```

In [10]:
```
#Let's check the same for test values
test_data.isnull().sum()
```

Out[10]:
```
geo_level_1_id                            0
geo_level_2_id                            0
```

```
geo_level_3_id                            0
count_floors_pre_eq                       0
age                                       0
area_percentage                           0
height_percentage                         0
land_surface_condition                    0
foundation_type                           0
roof_type                                 0
ground_floor_type                         0
other_floor_type                          0
position                                  0
plan_configuration                        0
has_superstructure_adobe_mud              0
has_superstructure_mud_mortar_stone       0
has_superstructure_stone_flag             0
has_superstructure_cement_mortar_stone    0
has_superstructure_mud_mortar_brick       0
has_superstructure_cement_mortar_brick    0
has_superstructure_timber                 0
has_superstructure_bamboo                 0
has_superstructure_rc_non_engineered      0
has_superstructure_rc_engineered          0
has_superstructure_other                  0
legal_ownership_status                    0
count_families                            0
has_secondary_use                         0
has_secondary_use_agriculture             0
has_secondary_use_hotel                   0
has_secondary_use_rental                  0
has_secondary_use_institution             0
has_secondary_use_school                  0
has_secondary_use_industry                0
has_secondary_use_health_post             0
has_secondary_use_gov_office              0
has_secondary_use_use_police              0
has_secondary_use_other                   0
dtype: int64
```

In [11]:
```
train_labels.isnull().sum()
```

Out[11]:
```
damage_grade    0
dtype: int64
```

Great! None of the datset have any missing value. Let's now move on to Exploratory Data Analysis. First, let's see what kind of features do we have.

In [12]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 260601 entries, 802906 to 747594
Data columns (total 38 columns):
geo_level_1_id                            260601 non-null int64
geo_level_2_id                            260601 non-null int64
geo_level_3_id                            260601 non-null int64
count_floors_pre_eq                       260601 non-null int64
age                                       260601 non-null int64
```

```
area_percentage                           260601 non-null int64
height_percentage                         260601 non-null int64
land_surface_condition                    260601 non-null object
foundation_type                           260601 non-null object
roof_type                                 260601 non-null object
ground_floor_type                         260601 non-null object
other_floor_type                          260601 non-null object
position                                  260601 non-null object
plan_configuration                        260601 non-null object
has_superstructure_adobe_mud              260601 non-null int64
has_superstructure_mud_mortar_stone       260601 non-null int64
has_superstructure_stone_flag             260601 non-null int64
has_superstructure_cement_mortar_stone    260601 non-null int64
has_superstructure_mud_mortar_brick       260601 non-null int64
has_superstructure_cement_mortar_brick    260601 non-null int64
has_superstructure_timber                 260601 non-null int64
has_superstructure_bamboo                 260601 non-null int64
has_superstructure_rc_non_engineered      260601 non-null int64
has_superstructure_rc_engineered          260601 non-null int64
has_superstructure_other                  260601 non-null int64
legal_ownership_status                    260601 non-null object
count_families                            260601 non-null int64
has_secondary_use                         260601 non-null int64
has_secondary_use_agriculture             260601 non-null int64
has_secondary_use_hotel                   260601 non-null int64
has_secondary_use_rental                  260601 non-null int64
has_secondary_use_institution             260601 non-null int64
has_secondary_use_school                  260601 non-null int64
has_secondary_use_industry                260601 non-null int64
has_secondary_use_health_post             260601 non-null int64
has_secondary_use_gov_office              260601 non-null int64
has_secondary_use_use_police              260601 non-null int64
has_secondary_use_other                   260601 non-null int64
dtypes: int64(30), object(8)
memory usage: 69.6+ MB
```

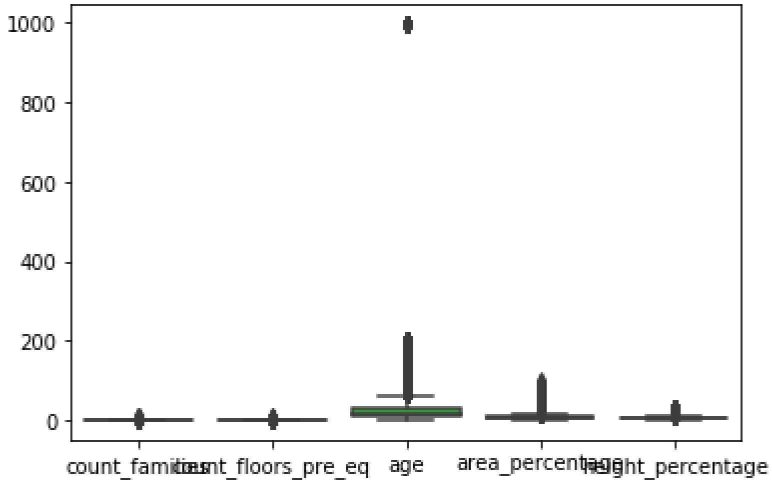
Most of our variables are factor or categorical variables. We will deal with them later. Let's seperate the numerical variables first.

In [13]:
```
numerical_features= ['count_families','count_floors_pre_eq', 'age', 'area_
percentage', 'height_percentage']
num_data= train_data[numerical_features]
```

In [15]:
```
#Let's visualize these numerical features
import seaborn as sns

sns.boxplot(data=num_data)
```

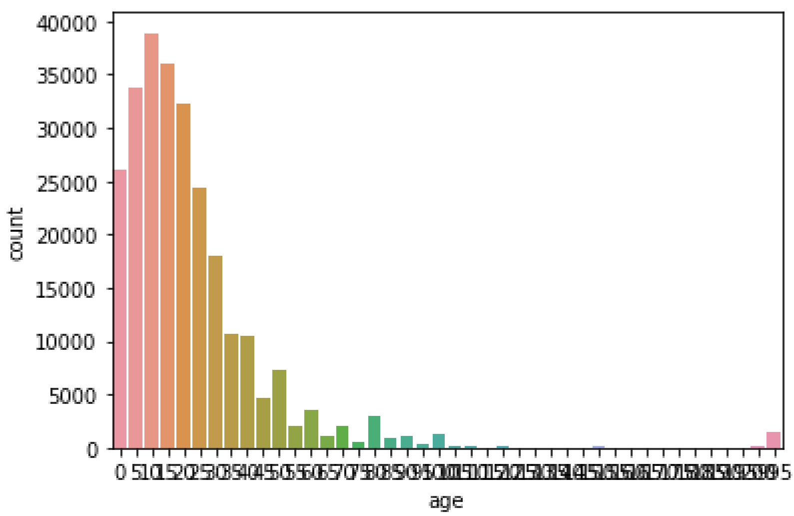
Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xd2e2f10>
```



We can see that most values are distributed from 0-200, except 'age' of the buildings which seems to have a few outliers. Let's take a look at it.

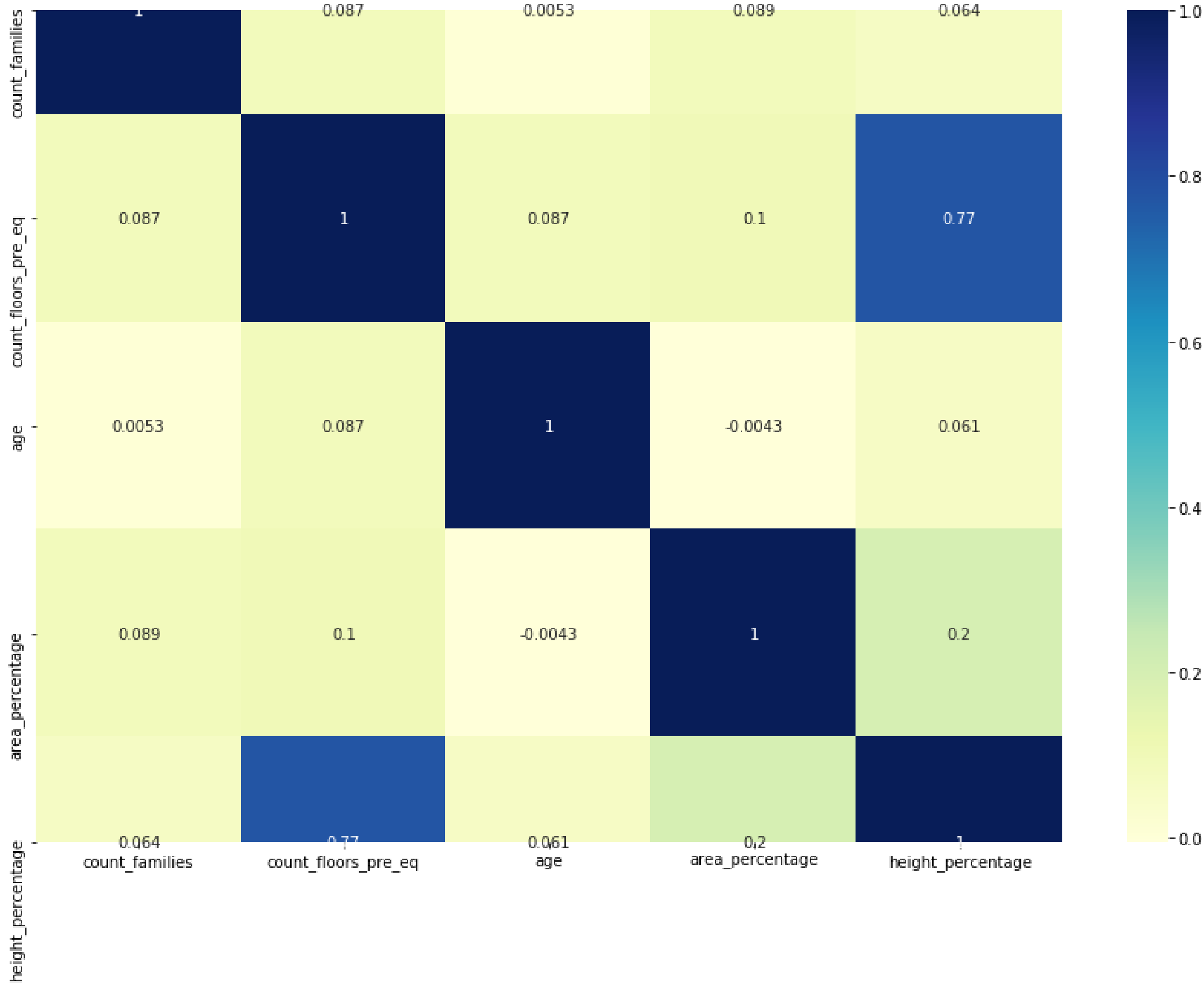
In [16]:
```
sns.countplot(num_data['age'])
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x113afa70>



We can see that most of the buildings are new. However, there are couple that are over 995 years old. Let's take a look at closer look at all numerical features.

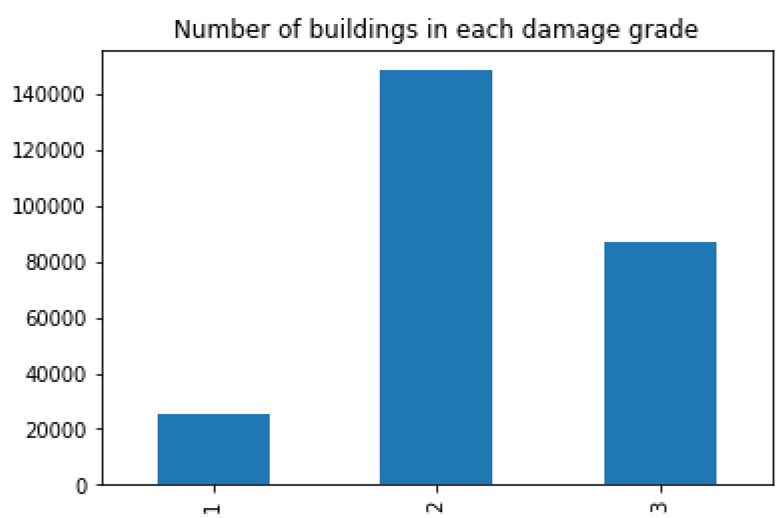
In [17]:
```
#Let's now see the correlation of all our numerical features.
plt.figure(figsize=(15,10))
correlation = num_data.corr()
sns.heatmap(correlation, annot= True, cmap="YlGnBu")
plt.show();
```



We can see that there is no significant correlation among all our numerical variables except 'count_floor_per_eq and 'height_percentage'. We will take care of that during our feature selection.

In [18]:
```
#lets's see our training data
(train_labels.damage_grade.value_counts().sort_index().plot.bar(title= 'Nu
mber of buildings in each damage grade'))
```

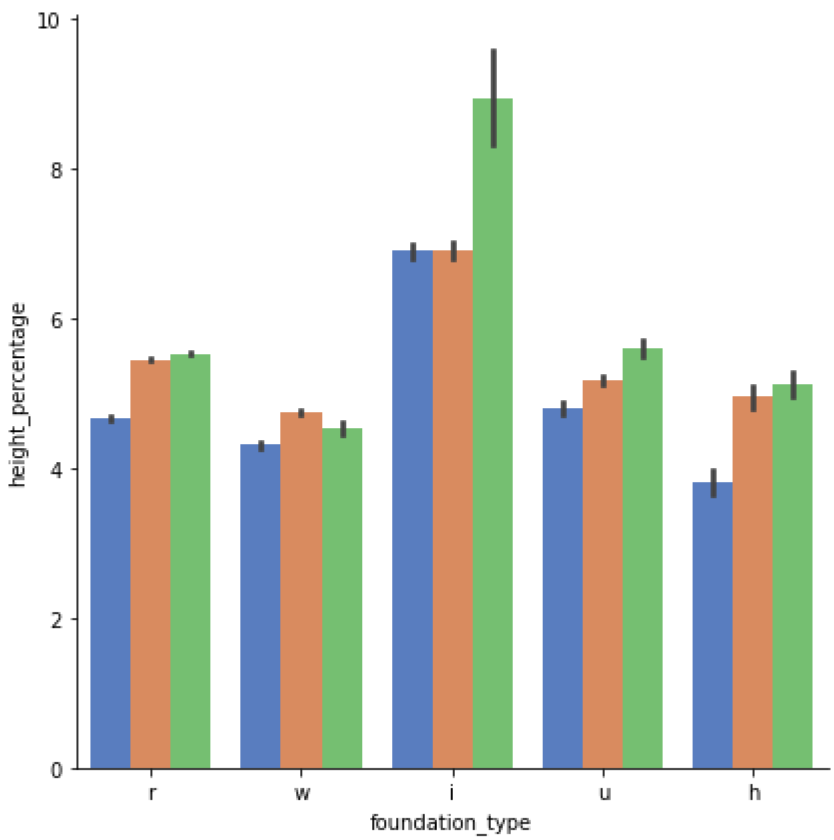
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0xd503670>



Looks like most buildings had medium damage.

In [19]:
```
#Let's analyze our categorical variables
merged_data= pd.merge(train_data, train_labels, on= 'building_id')
# Set up a factorplot
g = sns.factorplot("foundation_type", "height_percentage", "damage_grade",
 data=merged_data, kind="bar", size=6, palette="muted", legend=False)
plt.show()
```

```
c:\users\rakbansal\appdata\local\programs\python\python37-32\lib\site-pack
ages\seaborn\categorical.py:3666: UserWarning: The `factorplot` function h
as been renamed to `catplot`. The original name will be removed in a futur
e release. Please update your code. Note that the default `kind` in `facto
rplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
c:\users\rakbansal\appdata\local\programs\python\python37-32\lib\site-pack
ages\seaborn\categorical.py:3672: UserWarning: The `size` paramter has bee
n renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



In [20]:
```
#Let's do some feature selection. Let's run a Random Forest to get feature
 importances.
train_label= train_labels[['damage_grade']]
```

In [21]:
```
#Coding categorical Data
train_dummy= pd.get_dummies(train_data)
test_dummy= pd.get_dummies(test_data)
```

In [22]:
```
#Creating a Random Forest Model
```

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=10, random_state=5, max_depth=1
0)
```

In [23]:
```
#Fitting the model on Test Data
model.fit(train_dummy,train_label)
```

```
c:\users\rakbansal\appdata\local\programs\python\python37-32\lib\site-pack
ages\ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was
 passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
```

Out[23]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=5, verbos
e=0,
                      warm_start=False)
```

Let's Take a look at 'Feature Importances' of our model based on which we will select the most significant features.

In [29]:
```
#Feature importance in Descending order numbers
feature_importances = pd.Series(model.feature_importances_, index= train_d
ummy.columns).sort_values(ascending=False)
print(importances)
```

```
geo_level_1_id                            0.435828
foundation_type_r                         0.302499
has_superstructure_mud_mortar_stone       0.063477
geo_level_2_id                            0.056666
age                                       0.021128
foundation_type_i                         0.017616
has_superstructure_stone_flag             0.013464
has_superstructure_cement_mortar_brick    0.013269
geo_level_3_id                            0.012858
other_floor_type_q                        0.005916
area_percentage                           0.005730
count_families                            0.005020
height_percentage                         0.004641
roof_type_q                               0.004253
has_superstructure_mud_mortar_brick       0.003993
roof_type_n                               0.003654
has_superstructure_timber                 0.002421
count_floors_pre_eq                       0.002092
has_secondary_use                         0.001867
foundation_type_h                         0.001857
roof_type_x                               0.001836
has_superstructure_adobe_mud              0.001509
other_floor_type_x                        0.001503
ground_floor_type_v                       0.001450
has_superstructure_rc_non_engineered      0.001407
ground_floor_type_f                       0.001140
```

```
has_superstructure_other                     0.001038
foundation_type_u                            0.001028
position_s                                   0.001024
has_superstructure_cement_mortar_stone       0.000843
                                               ...
other_floor_type_j                           0.000386
other_floor_type_s                           0.000364
has_superstructure_rc_engineered             0.000352
legal_ownership_status_v                     0.000340
position_j                                   0.000340
position_t                                   0.000270
plan_configuration_d                         0.000244
legal_ownership_status_a                     0.000236
ground_floor_type_m                          0.000218
ground_floor_type_z                          0.000206
legal_ownership_status_r                     0.000196
foundation_type_w                            0.000185
plan_configuration_q                         0.000184
plan_configuration_u                         0.000167
legal_ownership_status_w                     0.000139
has_secondary_use_industry                   0.000138
has_secondary_use_rental                     0.000118
position_o                                   0.000041
plan_configuration_a                         0.000027
plan_configuration_s                         0.000026
has_secondary_use_institution                0.000015
has_secondary_use_school                     0.000010
has_secondary_use_health_post                0.000008
has_secondary_use_gov_office                 0.000003
plan_configuration_o                         0.000002
plan_configuration_c                         0.000001
plan_configuration_f                         0.000000
plan_configuration_m                         0.000000
plan_configuration_n                         0.000000
has_secondary_use_use_police                 0.000000
Length: 68, dtype: float64
```

In [35]:

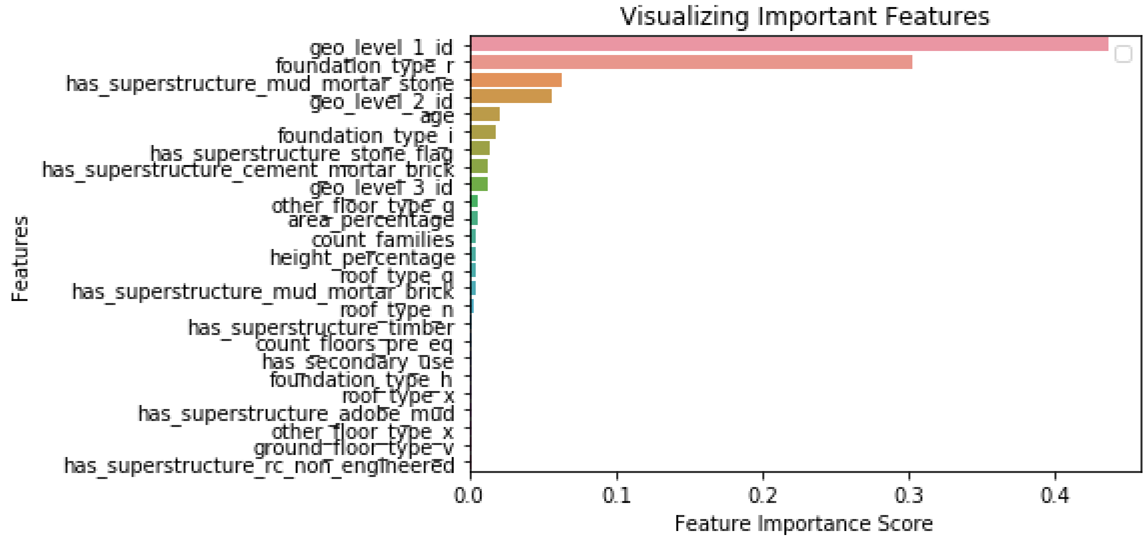
```
#Let's Visualize the same
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
feature_importances= feature_importances.nlargest(25) #Selects tehe top 25
 features
sns.barplot(x=feature_importances, y=feature_importances.index)
# Add labels to your graph

plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
```

```
No handles with labels found to put in legend.
```

Out[35]:

```
<matplotlib.legend.Legend at 0x1b0025b0>
```



In [39]:
```
#Creating a list of our newly selected top 25 Features
new_features= list(feature_importances.index)
```

In [40]:
```
#Initializing our training and testing data with new features
training_dataset= train_dummy[new_features]
testing_dataset= test_dummy[new_features]
```

In [43]:
```
#Let's take a look at new training dataset
training_dataset.head()
```

Out[43]:

	geo_level_1_id	foundation_type_r	has_superstructure_mud_mortar_stone	geo_level_2_id
building_id				
802906	6	1	1	487
28830	8	1	1	900
94947	21	1	1	363
590882	22	1	1	418
201944	11	1	0	131

5 rows × 25 columns

Let us now divide the training dataset into 'Training data' and 'validation data' before we apply it on the 'test data'.

In [49]:
```
#Dividing training data into train and validation dataset
from sklearn.model_selection import train_test_split
X= training_dataset
y= train_label
```

In [50]:
```
# Split dataset into training set and validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3) # 7
```

```
0% training and 30% test
```

In [70]:
```
# Create the model with 200 trees
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=200, oob_score = True, n_jobs
= -1, random_state =10, max_features = "auto", min_samples_leaf = 50)
```

In [71]:
```
#Let;s fit the model on training dataset
trained=model.fit(X_train, y_train)
```

```
c:\users\rakbansal\appdata\local\programs\python\python37-32\lib\site-pack
ages\ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was
 passed when a 1d array was expected. Please change the shape of y to (n_s
amples,), for example using ravel().
```

In [72]:
```
#Predict on Validation dataset
predicted= model.predict(X_val)
```

In [63]:
```
#Test the accuracy of the Model- Import required libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn import metrics
```

In [73]:
```
print("Accuracy:",metrics.accuracy_score(y_val, predicted))
```

```
Accuracy: 0.6918049142374746
```

In [56]:
```
from sklearn.metrics import classification_report
```

In [74]:
```
print (classification_report(y_val, predicted) )
```

```
              precision    recall  f1-score   support

           1       0.66      0.34      0.45      7546
           2       0.68      0.89      0.77     44580
           3       0.75      0.46      0.57     26055

    accuracy                           0.69     78181
   macro avg       0.70      0.56      0.60     78181
weighted avg       0.70      0.69      0.67     78181
```

The model provides an accuracy of 0.69

In [75]:
```
#Let's make predictions on the test dataset
damage_predicted= model.predict(testing_dataset)
```

In [78]:
```
print(damage_predicted)
```

```
[3 2 2 ... 2 2 2]
```

In [80]:
```
damage_predicted.shape
```

Out[80]: (86868,)

In [ ]:

```

```